SUMMARY:

Our project has been going along well, but we’ve made it just shy of our checkpoint goals. By the checkpoint deadline, we were planning on getting the plane detection and tracking done serially, and begin working to make our plane detection in parallel as well. We’ve also begun work on the sequential renderer. This process was sped up drastically by the starter code for tracking we found by Kyle Hounslove. We don’t yet have the parallel detection completed as of yet, but the way we’ve been going about is thusfar is modifying the findContours function in the build of OpenCV we’re using.

A bottleneck for our project has been the issue of appropriate setup. While Joe was able to set up a working C++ workspace and OpenCV to test the implementation of a plane detector and tracker, he also spent a while trying to build OpenCV from source on the latedays cluster. Prof. Railing informed them that the highest sourced implementation of OpenCV that could be built on the clusters was 2.0.0 (not the 3.3.1 implementation we needed), and that later implementations required CUDA 6.0 or greater, which couldn’t be installed, and required permissions which we as students didn’t have access to. We created an Redhat AWS workspace and begun trying to install OpenCV and all of the dependencies required. However, we begun to get problems for the video-capture software ffmpeg, and even after re-installing the ffmpeg and re-building OpenCV with it clearly in the build-path, OpenCV never recognized it. Finally, we decided to work with an Ubuntu 16.04 instance with 16 CPU’s, and the installation of both OpenCV and OpenGL went extremely smoothly!

On this workspace, we have compiled and run our sequential tracking and plane-detection algorithm, and have begun to parallelize the tracking portion. The tracking/plane-detection works by taking in a .avi file, and then processing the video by checking sequential frames for movement in the figure. If there’s any significant movement in pixels, a threshold image is drawn up, and the center of mass is calculated of this movement is calculated and tracked. We based this implementation off of Kyle’s work on the subject, but have been working on further improving its speed by parallelize the tracking of movement.

DELIVERABLES:

We actually plan to deliver on most all of the deliverables on our project page! Our exception is the mobile-device application testing. This is because he headache that setup required caused a large overhead on our overall workflow, and we’d rather not have to deal with that when porting to a mobile device. However, we do plan to port this over to a workspace with available GPU’s and test speedup utilizing GPU’s, rather than just a large amount of CPU’s.

GOALS:

We plan to hit all of our goals! We might be slightly lagging behind from where we wanted to be at this point, but our concise schedule has small enough goals that we plan to implement.

SCHEDULE:
Below is a chart to further detail the work we plan to complete over the coming weeks.

<table>
<thead>
<tr>
<th>11/21-11/24</th>
<th>11/24-11/27</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelize the findContours function to implement parallelization in plane-detection (JOE)</td>
<td>Complete a point-based renderer (TOYA).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11/27-12/1</th>
<th>12/1-12/5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look into feasibility of expanding application to work on GPU. (JOE)</td>
<td>Expand renderer to include more complex shapes (TOYA).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12/5-12/8</th>
<th>12/8-12/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelize renderer (JOE AND TOYA).</td>
<td>Optimize parallelization, tweak parameters, present. (JOE AND TOYA)</td>
</tr>
</tbody>
</table>

Poster Session:

We plan to present a video of our tracker and renderer at work, a graph of the speedup when utilizing different amount of threads, and a poster detailing the work we completed and details of our project. Our stretch goal would be comparing CPU-only implementation to a GPU-based implementation of our project.

Issues and Concerns:

We don’t have many issues we see yet, outside of the large amount of work we have in terms of coding.
Project Proposal:

Name:
Victoria Rosuello (vdr)
Joseph Chartouni (jchartou)

URL:
http://www.andrew.cmu.edu/user/jchartou/

SUMMARY:
We plan to create an optimized surface detection and renderer algorithm to render objects on a
seen plane in real time. We are initially going to create this optimized implementation on the GPU. If
we succeed, we will also complete one for multi-core CPU platforms, and then perform an analysis
on both systems’ performance characteristics to compare and contrast them. While we plan to
initially implement this on a real-time computer, we hope to also be able to create an application
runnable on a mobile devices as well.

BACKGROUND:
Our algorithm can be broken up into three distinct portions. The first part is the actual detection of
a flat plane in a given image that could be drawn upon. The second would be the tracking of the said
plane over real-time applications. The final part would be the rendering of a given item on the given
plane. Each of these parts would be initially implemented sequentially, and then done so in parallel.
For the detection of a plane, we’re planning on using OpenCV to extract features, compute the
homographies of the features using the RANSAC method, and create our planed surface to sketch on
(the plane detection of our algorithm). To continually track this plane, we’re planning on utilizing a
Lucas-Kanade Tracker to follow our object once it’s been instantiated. Finally, for the rendering of
our object, we’re planning on using OpenGL to render these images on the planes.
Parallelization could definitely improve parts of our algorithm! For example, the feature extraction
and matching between our imposed plane and the image we gather could be parallelized over
multiple processors.

THE CHALLENGE:
This algorithm provides a lot of challenges that we hope to overcome. First among them is the
transition from sequential code to parallel code. There are some steps (such as the detection of a
plane), which seem very easy to do sequentially. The calculation of an available plane uses
arbitrarily swaths of memory, and using a sequential algorithm places all of that needed memory in
a single processor. However, when parallelizing the task, the given plane we’re looking for could
have be processed by multiple processors, making this problem of workload and memory sharing an issue. This makes mapping the workload of our space extremely challenging, as we need to balance utilizing multiple processors with divvying up the workspace efficiently.

RESOURCES:
To test our implementation, we will be harvesting video footage of real-time video and then running the code on the late-days cluster. We currently do not plan to use a code base, and are planning on starting from scratch. So far, we’ve looked into using OpenCV for an object-detection library. We’re planning on using this to extract features, compute homographies using the RANSAC method, and create our planed surface to sketch on (the plane detection of our algorithm), as well as the implementation of our Lucas-Kanade Tracker. The OpenGL functions that we plan to use take care of a large part of the rendering we have, and we can set the center point of this rendering to be the center point of the plane we have.

GOALS AND DELIVERABLES.
In order to achieve a “successful project”, we must implement the baseline algorithm described above in parallel that can run efficiently on a GPU. This means writing a parallel plane detection algorithm, which can be tracked by a parallel tracking algorithm, and finally be drawn upon with a parallel rendering algorithm. In order to go above and beyond the scope of our project, we could implement this algorithm on multiple CPUs and perform analytics to compare how the program runs on both platforms. We could also try porting our code onto a mobile device and seeing whether or not it functions properly and efficiently. In case the work goes slower than initially anticipated, we could probably utilize a sequential portion of the algorithm such as plane detection or tracking.
We hope to see quite a bit of speedup between the sequential and parallel versions of our code! We can’t come up with a decent goal as of now, but we believe it’s fair to assume at least 5x speedup.
To perform a demo of our project during the poster session, we could take real time video, upload it to a late-days cluster, perform our algorithm, and then get back a video of the object being rendered on a plane in real time! We will be able to show up speedup graphs as well.

PLATFORM CHOICE.
We plan to use C++, as it has the support needed for OpenCV, and we have experience creating parallelized code in the language.

SCHEDULE.
Current Schedule:

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential planar finding</td>
<td>Parallel planar finding</td>
<td>Sequential planar tracking</td>
<td>Parallel planar tracking</td>
<td>Sequential Rendering</td>
<td>Parallel Rendering</td>
</tr>
</tbody>
</table>